NATIONAL RESEARCH UNIVERSITY

HIGHER SCHOOL OF ECONOMICS

SCHOOL OF BUSINESS INFORMATICS

PROJECT PROPOSAL

DESIGN AND DEVELOPMENT OF DATABASE FOR A SKI BASE

ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ

ГОРНОЛЫЖНОЙ БАЗЫ

Storozhok Maria, group 1908

Supervisor:

Assistant professor,

Yampolsky Sergey Mikhailovich

MOSCOW

2020

**Оглавление**

[Постановка задачи 3](#_Toc74253201)

[Использованные инструменты 4](#_Toc74253202)

[Описание предметной области 4](#_Toc74253203)

[Проектирование базы данных 5](#_Toc74253204)

[Инфологическая модель предметной области 5](#_Toc74253205)

[Описание сущностей предметной области 6](#_Toc74253206)

[Описание атрибутов предметной области с указанием первичных и внешних ключей таблиц 8](#_Toc74253207)

[Описание связей между сущностями предметной области 11](#_Toc74253208)

[Даталогическая модель данных 14](#_Toc74253209)

[Реализация базы данных в СУБД 15](#_Toc74253210)

[Запросы на языке SQL 21](#_Toc74253211)

[Пользовательские функции, представления, 22](#_Toc74253212)

[триггеры и хранимые процедуры 22](#_Toc74253213)

[Список используемой литературы 28](#_Toc74253214)

[Заключение 28](#_Toc74253215)

# Постановка задачи

***Целью*** моей курсовой работы являются проектирование и разработка базы данных для горнолыжной базы, которая будет отображать информацию о работе базы, включающую данные об оплатах посетителями спусков со склонов, расположенных на территории базы, и аренде/возврате инвентаря, являющегося имуществом базы.

В современном мире работу такой большой организации, соединяющей в своей деятельности столько компонентов, как горнолыжный курорт, в системе которого непрерывно циркулирует огромное количество информации, нельзя представить без баз данных, в которых хранится вся информация о посетителях, работниках и процессах, происходящих при ее работе. Эта причина являет собой ***актуальность выбранной мною темы***. ***Практическое значение*** состоит в том, что использование баз данных повышает эффективность работы горнолыжной базы, а вследствие и качество обслуживания, безопасность на склонах (за счет лучшего контроля за работой спасателей и охранников территории), что приводит к притоку новых клиентов, сокращению нежелательных расходов и увеличению прибыли.

Таким образом, ***объектом*** моего исследования является база данных лыжной базы, ***предметом*** - автоматизация и улучшение процессов хранения и записи данных о деятельности базы. Ввиду всего, написанного выше, возникают следующие ***задачи исследования***, определяющие основные этапы для достижения поставленной цели:

* Изучить структуру и этапы работы горнолыжной базы, а также участников процессов, протекающих в ней (я смотрела на базу «Бобровый лог»), и построить инфологическую модель предметной области;
* Исследовать принципы деятельности горнолыжной базы и построить даталогическую модель данных;
* Создать базу данных горнолыжной базы, привести примеры запросов к ней;
* Создать запросы, хранимые процедуры, функции и триггеры на языке SQL для реализованной базы данных;

## Использованные инструменты

При проектировании предметной области (ER diagram) я использовала ERDPlus, при построении даталогической модели данных - WORD. База данных разработана при помощи университетского сервера и клиента DBeaver.

## Описание предметной области

Разработанная база данных предназначена для систематизации, хранения и мгновенного доступа к информации о посетителях и работниках горнолыжного курорта, а также о всех их действиях, совершенных на территории курорта.

Использование базы данных помогает легко совершить выборку клиентов или работников по заданным условиям, просмотреть количество спусков определенного клиента, найти самый популярный среди посетителей склон для спусков (по наибольшему количеству оплат того или иного склона), выявить наиболее трудолюбивых сотрудников, которые реже остальных берут отгулы, просмотреть всех работников горнолыжной базы и т.д. .

# Проектирование базы данных

***Методика проектирования базы данных*** – последовательность действий, позволяющая реально существующий объект предметной области представить в машинном виде.

Таким образом, процесс проектировки состоит из следующих этапов:

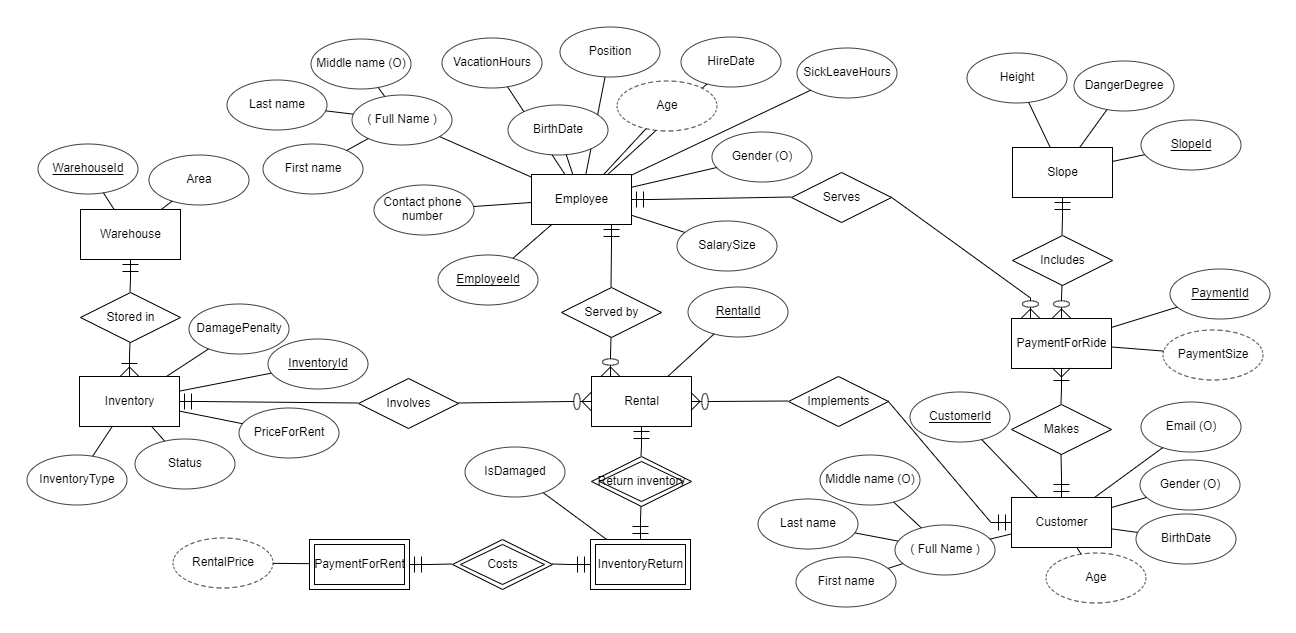
* Описание предметной области (уже сделали)
* Построение инфологической модели
* Выбор СУБД и создание даталогической модели
* Проектирование физической модели

Итак, следующая стадия – инфологическое проектирование.

## Инфологическая модель предметной области

***Инфологическая модель предметной области (Entity-Relationship Model, ER-модель)*** обеспечивает первоначальное описание информационного содержания автоматизируемых процессов (предметной области), согласовывая и объединяя в себе представления всех категорий пользователей.

Инфологическая модель базы данных горнолыжного курорта имеет следующий вид:



*Рисунок 1 – Схема инфологической модели предметной области*

Данная модель была построена с использованием ресурса ERDPlus, основанного на ***методологии Питера Чена*** (и использующего ***нотацию «вороньи лапки»***).

### Описание сущностей предметной области

Для всестороннего описания деятельности горнолыжной базы и агентов, принимающих в ней участие, мне понадобились 9 сущностей.

Таблицы в базах данных, которые будут описывать эти самые сущности, делятся на ***справочники и документы***. Справочники содержат статичную информацию, т.е. содержат сведения об объектах, субъектах и связях. Документы, в свою очередь, формируются на основе справочников и включают «интерактивные» элементы, т.е. хранят информацию о действиях, процессах, событиях и т.д. (данные по взаимодействию объектов/субъектов).

Сущности, описываемые таблицами-справочниками:

* Employee – данные о сотрудниках, необходимые для организации их работы на базе (предоставления им рабочего места);
* Slope – справочные данные о склонах, находящихся на территории базы;
* Customer – необходимые сведения о посетителях базы (для регистрации их в системе, предоставления им возможности оплаты спусков со склонов);
* Inventory – сведения об имеющемся на базе инвентаре, предназначенном для сдачи его напрокат;
* Warehouse – сведения о складах на территории горнолыжной базы, предназначенных для хранения инвентаря

Сущности, описываемые таблицами-документами:

* PaymentForRide – информация об оплатах посетителями своих спусков со склонов (каждый спуск конкретного клиента с конкретного склона оплачивается отдельно);
* Rental – задокументированная информация об аренде элемента инвентаря конкретным пользователем;
* InventoryReturn – информация о возврате конкретного арендованного предмета инвентаря, включающая факт его порчи либо возврата данного элемента инвентаря в сохранности;
* PaymentForRent – сведения об оплате конкретного проката (аренды инвентаря);

В реальной жизни счет, выставляемый за аренду инвентаря, воплощается в виде чека и ***является отдельной сущностью*** (зависящей от конкретной аренды).

### Описание атрибутов предметной области с указанием первичных и внешних ключей таблиц

Описание, которое поможет лучше понять специфику предметной области и работы горнолыжного курорта в целом, а также принципы устройства базы данных, которую предстоит сконструировать, представлено в таблице:

*Таблица 1 – Описание атрибутов сущностей предметной области*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сущность | Поля | Тип данных | Описание поля | Комментарий |
| Employee | EmployeeId | int | Идентификатор сотрудника | РК, суррогатный первичный ключ |
| Full name | varchar(60) | ФИО сотрудника | Обязательное (составное) поле |
| Contact phone number | bigint | Телефонный номер сотрудника | Обязательное поле |
| Gender | char(1) | Пол сотрудника | Необязательное поле, «ж», «м» или пустое значение (null) |
| BirthDate | date | Дата рождения сотрудника | Обязательное поле |
| Age | int | Возраст сотрудника | Обязательное (вычислимое) поле |
| Position | varchar(20) | Должность сотрудника | Обязательное поле |
| SalarySize | int | Размер заработной платы сотрудника | Обязательное поле, >30000 руб. |
| HireDate | date | Дата приема сотрудника на работу | Обязательное поле |
| VacationHours | int | Часы отпуска сотрудника | Обязательное поле |
| SickLeaveHours | int | Часы отпуска по болезни сотрудника | Обязательное поле |
| Slope | SlopeId | int | Номер склона | РК, первичный ключ |
| DangerDegree | int | Уровень опасности | Обязательное поле |
| Height | int | Высота склона | Обязательное поле |
| Customer | CustomerId | int | Идентификатор посетителя (каждый гость обязательно регистрируется при входе на базу) | РК, суррогатный первичный ключ |
| Full name | varchar(60) | ФИО посетителя | Обязательное (составное) поле |
| Gender | char(1) | Пол посетителя | Необязательное поле, «ж», «м» или пустое значение (null) |
| BirthDate | date | Дата рождения посетителя | Обязательное поле |
| Age | int | Возраст посетителя | Обязательное (вычислимое) поле |
| Email | varchar(50) | Адрес электронной почты посетителя | Необязательное поле |
| Inventory | InventoryId | int | Идентификатор элемента инвентаря | РК, суррогатный первичный ключ |
| InventoryType | varchar(20) | Тип инвентаря | Обязательное поле |
| PriceForRent | int | Цена аренды | Обязательное поле |
| DamagePenalty | int | Величина штрафа за ущерб (в рамках предметной области то же самое, что и величина залога, который в случае порчи инвентаря не возвращается) | Обязательное поле |
| Status | varchar(10) | Статус элемента инвентаря | Обязательное поле, “Available”, “Rented” или “Broken”, по умолчанию - “Available” |
| Warehouse | WarehouseId | int | Номер корпуса | РК, первичный ключ |
| Area | float | Площадь склада | Обязательное поле |
| PaymentForRide | PaymentId | int | Идентификатор оплаты спуска со склона | РК, суррогатный первичный ключ |
| PaymentSize | int | Цена спуска со склона | Обязательное поле, 300 – если посетитель не достиг 14 лет, 700 – если посетителю 14 лет и больше. Вычисляется по возрасту покупателя, оплачивающего спуск со склона. |
| Rental | RentalId | int | Идентификатор аренды | РК, суррогатный первичный ключ |
| InventoryReturn | IsDamaged | int | Идентификатор фиксирования факта порчи арендованного инвентаря | Обязательное поле, 1 –арендованный элемент инвентаря испорчен, 0 – не испорчен |
| PaymentForRent | RentalPrice | int | Цена аренды (к PriceForRent соответствующего типа инвентаря прибавляется величина залога, в случае если инвентарь был испорчен) | Обязательное (вычислимое) поле |

\* если бы мы заполняли таблицы на русском, то в самой бд нужно было бы установить тип nvarchar(max), а не varchar(max); при описании типов выше, в любом случае, достаточно указать общий тип varchar(max), не уточняя кодировку

### Описание связей между сущностями предметной области

Ниже приведена детальная информация о связях между обозначенными выше сущностями:

*Таблица 2 – Описание связей между сущностями предметной области*

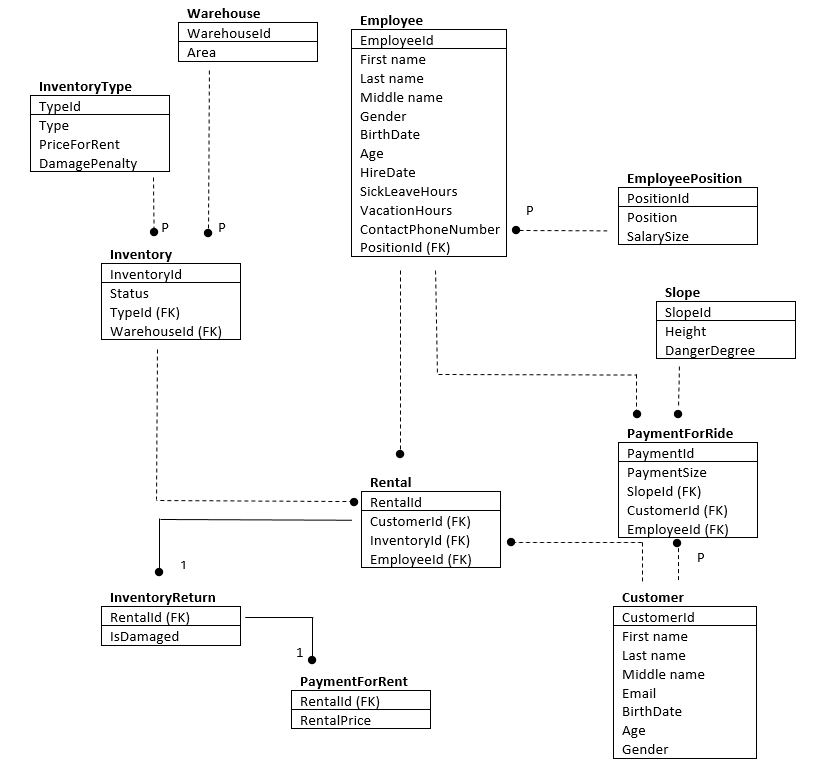
|  |  |  |  |
| --- | --- | --- | --- |
| Связь | Отношение | Главная таблица | Дочерняя таблица |
| Served by | Один ко многим | Employee | Rental |
| Serves | Один ко многим | Employee | PaymentForRide |
| Includes | Один ко многим | Slope | PaymentForRide |
| Makes | Один ко многим | Customer | PaymentForRide |
| Implements | Один ко многим | Customer | Rental |
| Stored in | Один ко многим | Warehouse | Inventory |
| Involves | Один ко многим | Inventory | Rental |
| Return inventory | Один к одному | Rental | InventoryReturn |
| Costs | Один к одному | InventoryReturn | PaymentForRent |

Теперь нужно уточнить, что именно означают данные связи:

1. Связь ***«Served by»*** символизирует тот факт, что оформлением аренды должен заниматься какой-то сотрудник. Эта связь неидентифицирующая, определена отношением «один ко многим», так как конкретный сотрудник может обслуживать аренду элементов инвентаря много раз, в то время как каждая аренда обслуживается лишь одним конкретным сотрудником. При этом существует необязательность связи со стороны сотрудника, так как не каждый сотрудник может работать в пункте проката (сотрудник может работать в прокате только в том случае, когда его должность – ‘RentalWorker’; это логично и будет всегда выполняться, так как охранник, например, никогда не придет в пункт проката, не встанет на кассу и не начнет выполнять чужую работу, а не свою).
2. Связь ***«Serves»*** показывает, что в процессе оплаты посетителем спуска со склона также должен участвовать какой-то сотрудник. Связь не является идентифицирующей и определяется отношением «один ко многим», так как конкретный сотрудник может обслуживать оплату спуска со склона много раз, но каждая такая оплата обслуживается лишь одним конкретным сотрудником. Пустые значения разрешены, так как не каждый сотрудник может обслуживать оплату спуска посетителем (сотрудник может заниматься данной деятельностью, когда его должность – ‘Caretaker’).
3. Связь ***«Includes»*** говорит нам о том, спуск с какого склон был оплачен конкретным покупателем при конкретной оплате. Является неидентифицирующей, определяется отношением «один ко многим», потому что конкретный склон могут оплатить множество раз, но при конкретной оплате оплачивается один конкретный склон. Склон может находиться в разработке и только готовиться к открытию для гостей горнолыжной базы, однако быть заранее занесенном в таблицу, на этот случай разрешены null-значения.
4. Связь ***«Makes»*** - это связь между посетителем и оплачиваемыми им спусками со склонов. Связь неидентифицирующая, определена также отношением «один ко многим», ведь посетителем могут быть оплачены много спусков, но оплачивает каждый конкретный спуск определенный покупатель. Пустые значения запрещены, так как ни посетитель не может не оплатить ни одного спуска (когда он в первый раз посещает базу, его просят заполнить анкету, таким образом, он попадает в базу данных, было бы странно, если бы после этого он, ни разу не прокатившись, ушел бы), ни оплата не может существовать сама по себе без того, кто ее совершит.
5. Связь ***«Implements»*** говорит о том, что какой-то посетитель взял напрокат какой-то элемент инвентаря. Эта связь неидентифицирующая, она определена отношением «один ко многим», так как конкретный посетитель может неоднократно брать инвентарь в прокат, в то время как каждая аренда осуществляется конкретным посетителем. Пустые значения не запрещены, так как посетитель может и вовсе не арендовать инвентарь
6. Связь ***«Stored in»*** олицетворяет тот факт, что каждый элемент инвентаря хранится на конкретном складе. Данная связь является неидентифицирующей и определяется отношением «один ко многим», так как на каждом складе хранится по несколько элементов инвентаря, предмет инвентаря, в свою очередь, не может храниться в двух и более местах одновременно. Пустые значения не разрешаются. Нам нужно знать, на каком складе хранится конкретный инвентарь. И мы не можем строить склад, который будет пустовать, это невыгодно и неоправданно.
7. Связь ***«Involves»*** показывает отношение между инвентарем и конкретной арендой. Связь неидентифицирующая, ее определяет отношение «один ко многим», это объясняется тем, что один и тот же элемент инвентаря может арендоваться множество раз, но при конкретной аренде арендуется один конкретный элемент инвентаря. Инвентарь, по каким-то причинам, может ни разу не быть арендован, или он может храниться на складе, но пока не выдаваться в аренду, поэтому разрешены пустые значения.
8. Связь ***«Return inventory»*** - связь между арендой конкретного инвентаря и его возвратом (в рамках этой аренды). Связь идентифицирующая, описываемая отношением «один к одному», так как каждой аренде соответствует один возврат, который ею определяется. Пустые значения запрещены, так как каждой аренде должен соответствовать возврат арендованного в ней.
9. Связь ***«Costs»*** говорит о том, что каждая конкретная возвращенная аренда стоит определенной суммы денег. Связь идентифицирующая, описывается отношением «один к одному», ведь каждая аренда должна сколько-то стоить, поэтому каждый возврат аренды определяет цену получившейся аренды, которая складывается из цены самой аренды и величины невозвращенного залога в том случае, если инвентарь был поврежден.

## Даталогическая модель данных

Модель, строящаяся на даталогическом уровне проектирования базы данных, основывается на уже построенной ER-диаграмме, описывающей предметную область. Эта модель отображает данные, содержащиеся в базе данных, а также связи между этими данными.



*Рисунок 2 – Схема даталогической модели данных*

## Реализация базы данных в СУБД

Создадим базу данных горнолыжного курорта на сервере ВШЭ при помощи средства DBeaver. Она будет называться Ski\_Base\_2021.

Построим таблицы, изображенные на даталогической модели выше, при помощи следующего кода:

**CREATE** **TABLE** Slope

(

SlopeId **INT** **NOT** **NULL**,

DangerDegree **INT** **NOT** **NULL**,

Height **INT** **NOT** **NULL**,

**PRIMARY** **KEY** (SlopeId)

);

**CREATE** **TABLE** Warehouse

(

WarehouseId **INT** **NOT** **NULL**,

Area **INT** **NOT** **NULL**,

**PRIMARY** **KEY** (WarehouseId)

);

При создании некоторых таблиц я использовала автоинкремент, а также устанавливала ограничения на значения некоторых полей и значения по умолчанию.

**CREATE** **TABLE** InventoryType

(

TypeId **INT** **IDENTITY**(1, 1) **NOT** **NULL**,

[Type] **VARCHAR**(20) **NOT** **NULL**,

PriceForRent **INT** **NOT** **NULL**,

DamagePenalty **INT** **NOT** **NULL**,

**CONSTRAINT** constr\_on\_type\_name **UNIQUE**([Type]),

**PRIMARY** **KEY** (TypeId)

);

**CREATE** **TABLE** Inventory

(

InventoryId **INT** **IDENTITY**(1, 1) **NOT** **NULL**,

Status **VARCHAR**(10) **NOT** **NULL** **DEFAULT** 'Available' **CHECK** (Status **IN** ('Available', 'Rented', 'Broken')),

WarehouseId **INT** **NOT** **NULL**,

TypeId **INT** **NOT** **NULL**,

**PRIMARY** **KEY** (InventoryId),

**FOREIGN** **KEY** (WarehouseId) **REFERENCES** Warehouse(WarehouseId),

**FOREIGN** **KEY** (TypeId) **REFERENCES** InventoryType(TypeId)

);

**CREATE** **TABLE** EmployeePosition

(

PositionId **INT** **IDENTITY**(1, 1) **NOT** **NULL**,

[Position] **VARCHAR**(20) **NOT** **NULL**,

SalarySize **INT** **NOT** **NULL**,

**CONSTRAINT** constr\_on\_position\_name **UNIQUE**([Position]),

**CONSTRAINT** constr\_on\_salary\_size **CHECK**(SalarySize >30000),

**PRIMARY** **KEY** (PositionId)

);

Создадим таблицы Employee и Customer с использованием созданной (код приведен ниже) пользовательской функции Ski\_Base\_2021.dbo.[get\_age], вычисляющей возраст человека по дате его рождения:

**CREATE** **TABLE** Employee

(

EmployeeId **INT** **IDENTITY**(1, 1) **NOT** **NULL**,

First\_name **VARCHAR**(20) **NOT** **NULL**,

Last\_name **VARCHAR**(20) **NOT** **NULL**,

Middle\_name **VARCHAR**(20),

BirthDate **DATE** **NOT** **NULL**,

Age **AS** dbo.[get\_age](BirthDate),

HireDate **DATE** **NOT** **NULL**,

SickLeaveHours **INT** **NOT** **NULL**,

VacationHours **INT** **NOT** **NULL**,

Gender **CHAR**(1) **CHECK** (Gender **IN** ('f', 'm', **NULL**)),

ContuctPhoneNumber **BIGINT** **NOT** **NULL**,

PositionId **INT** **NOT** **NULL**,

**PRIMARY** **KEY** (EmployeeId),

**FOREIGN** **KEY** (PositionId) **REFERENCES** EmployeePosition(PositionId)

);

**CREATE** **TABLE** Customer

(

CustomerId **INT** **IDENTITY**(1, 1) **NOT** **NULL**,

First\_name **VARCHAR**(20) **NOT** **NULL**,

Last\_name **VARCHAR**(20) **NOT** **NULL**,

Middle\_name **VARCHAR**(20),

Email **VARCHAR**(50),

BirthDate **DATE** **NOT** **NULL**,

Age **AS** dbo.[get\_age](BirthDate),

Gender **CHAR**(1) **CHECK** (Gender **IN** ('f', 'm', **NULL**)),

**PRIMARY** **KEY** (CustomerId)

);

В таблице оплаты за спуск со склона (PaymentForRide) стоимость спуска (PaymentSize) вычисляется автоматически в зависимости от возраста клиента, для этого используется функция dbo.payment\_for\_ride\_size, код которой приведен в следующем разделе моей работы:

**CREATE** **TABLE** PaymentForRide

(

PaymentId **INT** **IDENTITY**(1, 1) **NOT** **NULL**,

SlopeId **INT** **NOT** **NULL**,

CustomerId **INT** **NOT** **NULL**,

EmployeeId **INT** **NOT** **NULL**,

PaymentSize **AS** dbo.payment\_for\_ride\_size(CustomerId),

**PRIMARY** **KEY** (PaymentId),

**FOREIGN** **KEY** (SlopeId) **REFERENCES** Slope(SlopeId),

**FOREIGN** **KEY** (CustomerId) **REFERENCES** Customer(CustomerId),

**FOREIGN** **KEY** (EmployeeId) **REFERENCES** Employee(EmployeeId)

);

Создадим последние таблицы, относящиеся к аренде инвентаря:

**CREATE** **TABLE** Rental

(

RentalId **INT** **IDENTITY**(1, 1) **NOT** **NULL**,

InventoryId **INT** **NOT** **NULL**,

CustomerId **INT** **NOT** **NULL**,

EmployeeId **INT** **NOT** **NULL**,

**PRIMARY** **KEY** (RentalId),

**FOREIGN** **KEY** (InventoryId) **REFERENCES** Inventory(InventoryId),

**FOREIGN** **KEY** (CustomerId) **REFERENCES** Customer(CustomerId),

**FOREIGN** **KEY** (EmployeeId) **REFERENCES** Employee(EmployeeId)

);

**CREATE** **TABLE** InventoryReturn

(

RentalId **INT** **NOT** **NULL**,

IsDamaged **INT** **NOT** **NULL** **CHECK** (IsDamaged = 0 **OR** IsDamaged = 1),

**PRIMARY** **KEY** (RentalId),

**FOREIGN** **KEY** (RentalId) **REFERENCES** Rental(RentalId)

);

**CREATE** **TABLE** PaymentForRent

(

RentalId **INT** **NOT** **NULL**,

RentalPrice **INT** **NOT** **NULL**,

**PRIMARY** **KEY** (RentalId),

**FOREIGN** **KEY** (RentalId) **REFERENCES** InventoryReturn(RentalId)

);

Заполним созданные таблицы данными:

**SET** **IDENTITY\_INSERT** Ski\_Base\_2021.dbo.EmployeePosition **ON**

**INSERT**

**INTO** Ski\_Base\_2021.dbo.EmployeePosition(PositionId, [Position], SalarySize)

**VALUES** (1, 'SupportWorker', 60000)

**SET** **IDENTITY\_INSERT** Ski\_Base\_2021.dbo.EmployeePosition **OFF**

**INSERT**

**INTO** Ski\_Base\_2021.dbo.EmployeePosition([Position], SalarySize)

**VALUES** ('Caretaker', 45000), ('SecurityGuard', 35000), ('RentalWorker', 35000), ('DoctorOnDuty', 70000), ('Cleaner', 47000), ('Repairer', 50000)

Изображение выглядит как текст

Автоматически созданное описание

*Рисунок 3 – Заполнение таблицы должностей*

Изображение выглядит как стол

Автоматически созданное описание

*Рисунок 4 – Заполненная таблица-справочник должностей*

**SET** **IDENTITY\_INSERT** Ski\_Base\_2021.dbo.InventoryType **ON**

**INSERT**

**INTO** Ski\_Base\_2021.dbo.InventoryType(TypeId, [Type], PriceForRent, DamagePenalty)

**VALUES** (1, 'Skis', 1000, 600)

**SET** **IDENTITY\_INSERT** Ski\_Base\_2021.dbo.InventoryType **OFF**

**INSERT**

**INTO** Ski\_Base\_2021.dbo.InventoryType([Type], PriceForRent, DamagePenalty)

**VALUES** ('SkiPoles', 700, 400), ('KneePads', 500, 200), ('ElbowPads', 500, 200), ('ProtectiveGloves', 400, 150), ('ProtectiveHelmet', 1300, 700),

('Snowboard', 1500, 1000)

Изображение выглядит как стол

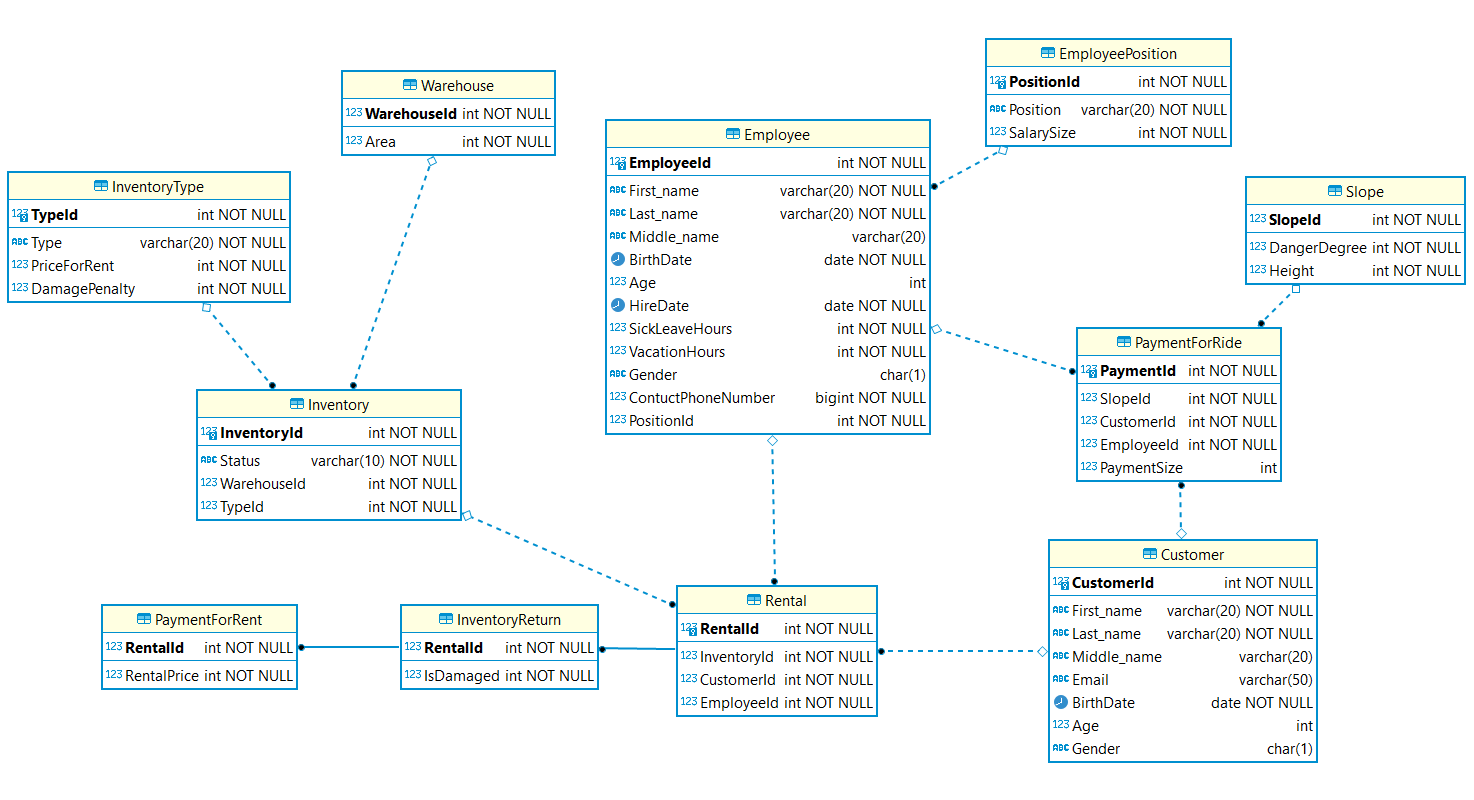
Автоматически созданное описание

*Рисунок 5 – Заполненная таблица-справочник типов инвентаря*

Аналогично заполним остальные таблицы.

Мы реализовали таблицы базы данных, заполнили их тестовыми данными.

Теперь можем посмотреть на диаграмму получившейся базы данных:



*Рисунок 6 – Диаграмма сконструированной базы данных в СУБД*

### Запросы на языке SQL

Теперь база данных реализована и готова к использованию. Покажем на примере разных запросов, чем она может быть полезна.

Запрос 1:

Рассмотрим первый запрос

Изображение выглядит как текст, стол

Автоматически созданное описание

*Рисунок 7 – Запрос 1*

При выполнении данного запроса на экране появится список с должностями и количеством новых сотрудников, пришедших на эти должности в этом календарном году. Список упорядочен по названиям должностей.

Запрос 2:

**SELECT** tab1.[Type], tab1.Amount Available, tab1.Amount Not\_Available

**FROM** (**SELECT** [Type], **COUNT**(\*) 'Amount'

**FROM** Inventory i **LEFT** **JOIN** InventoryType it

**ON** i.TypeId = it.TypeId

**WHERE** i.Status = 'Available'

**GROUP** **BY** [Type], Status) tab1

**LEFT** **JOIN**

(**SELECT** [Type], **COUNT**(\*) 'Amount'

**FROM** Inventory i **LEFT** **JOIN** InventoryType it

**ON** i.TypeId = it.TypeId

**WHERE** i.Status **NOT** **LIKE** 'Available'

**GROUP** **BY** [Type]) tab2

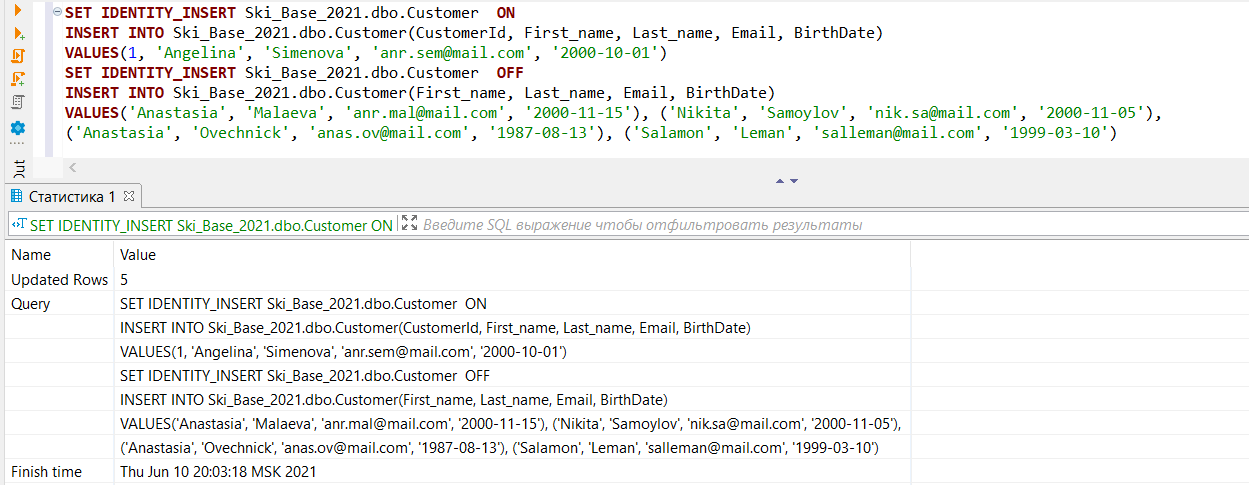
**ON** tab1.[Type] = tab2.[Type]

Изображение выглядит как текст

Автоматически созданное описание

Этот запрос показывает количество доступных и не доступных в данный момент элементов инвентаря каждого типа из всех имеющихся на базе.

Запрос 3:



В этом запросе мы добавляем новые записи в таблицу Посетитель (Customer).

### Пользовательские функции, представления,

### триггеры и хранимые процедуры

А теперь перейдем к тому, что и составляет функциональность нашей базы данных.

Первая пользовательская функция:

**CREATE** **FUNCTION** dbo.get\_age

(@birth **DATE**)

**RETURNS** **INT**

**AS** **BEGIN**

**DECLARE** @age **INT**

**DECLARE** @today **DATE**

**DECLARE** @yeardif **INT**

**DECLARE** @monthdif **INT**

**DECLARE** @daydif **INT**

**SET** @today = (**SELECT** **CONVERT**(**DATE**, **GETDATE**()))

**SET** @yeardif = (**SELECT** **YEAR**(@today) - **YEAR**(@birth))

**SET** @monthdif = (**SELECT** **MONTH**(@today) - **MONTH**(@birth))

**SET** @daydif = (**SELECT** **DAY**(@today) - **DAY**(@birth))

**IF** @monthdif < 0 **SET** @age = (@yeardif - 1)

**ELSE** **IF** @monthdif >= 0 **AND** @daydif < 0 **SET** @age = (@yeardif - 1)

**ELSE** **SET** @age = @yeardif

**RETURN** @age

**END**

Данная функция вычисляет возраст (используется для таблиц Сотрудник и Посетитель) индивида, получая на вход лишь дату его рождения.

Вторая пользовательская функция:

**CREATE** **FUNCTION** dbo.payment\_for\_ride\_size

(@cid **INT**)

**RETURNS** **INT**

**AS**

**BEGIN**

**DECLARE** @payment **INT**

**DECLARE** @age **INT**

**SET** @age = (**SELECT** Age **FROM** Ski\_Base\_2021.dbo.Customer **WHERE** CustomerId = @cid)

**IF** @age < 14 **SET** @payment = 300

**ELSE** **SET** @payment = 700

**RETURN** @payment

**END**

Вычисляет цену спуска со склона для конкретного клиента, так как цена рассчитывается из соображения, достиг ли человек 14-летия.

Третья пользовательская функция:

**CREATE** **FUNCTION** dbo.return\_employee\_position

(@eid **INT**)

**RETURNS** **VARCHAR**(20)

**AS** **BEGIN**

**DECLARE** @pos **VARCHAR**(20)

**SET** @pos = (**SELECT** [Position] **FROM** employees\_positions ep **WHERE** ep.EmployeeId = @eid)

**RETURN** @pos

**END**

Возвращает должность сотрудника, чей идентификатор был подан ей на вход.

Первое представление:

**CREATE** **VIEW** dbo.joined\_tables **AS**

**SELECT** InventoryId, [Type], PriceForRent, DamagePenalty, Status

**FROM** InventoryType **RIGHT** **JOIN** Inventory

**ON** InventoryType.TypeId = Inventory.TypeId

Оно упростит код для моих триггеров.

Возвращает список строк идентификаторов имеющихся на базе элементов инвентаря вместе с соответствующими им ценой за аренду, штрафом за повреждения и статусом на момент использования представления.

Второе представление:

**CREATE** **VIEW** employees\_positions **AS**

**SELECT** EmployeeId, Employee.PositionId, [Position]

**FROM** Employee **LEFT** **JOIN** EmployeePosition

**ON** Employee.PositionId = EmployeePosition.PositionId

Возвращает список сотрудников (их идентификаторов) вместе с должностями, которые они занимают.

Первый триггер:

При добавлении новой аренды в таблицу Аренда вызывает данный триггер, меняющий статус вновь арендованного инвентаря на ‘Rented’, так как больше он не свободен и в ближайшее время его не смогут арендовать другие желающие.

**CREATE** **OR** **ALTER** **TRIGGER** change\_inventory\_status\_after\_rent **ON** Ski\_Base\_2021.dbo.Rental

**AFTER** **INSERT**

**AS** **BEGIN**

**UPDATE** dbo.Inventory **SET** Status = 'Rented'

**WHERE** Inventory.InventoryId = (**SELECT** InventoryId **FROM** inserted)

**END**;

Второй триггер:

**CREATE** **TRIGGER** insert\_payment\_for\_rent **ON** Ski\_Base\_2021.dbo.InventoryReturn

**AFTER** **INSERT**

**AS** **BEGIN**

**INSERT** **INTO** dbo.PaymentForRent **VALUES**

(

(**SELECT** RentalId **FROM** inserted),

((**SELECT** PriceForRent **FROM** Ski\_Base\_2021.dbo.joined\_tables

**WHERE** InventoryId = (**SELECT** InventoryId **FROM** Rental **WHERE** Rental.RentalId = (**SELECT** RentalId **FROM** inserted))) + (

(**SELECT** IsDamaged **FROM** inserted) \* (**SELECT** DamagePenalty

**FROM** Ski\_Base\_2021.dbo.joined\_tables Inv **INNER** **JOIN** Rental

**ON** Rental.InventoryId = Inv.InventoryId

**WHERE** Rental.RentalId = (**SELECT** RentalId **FROM** inserted)))

))

**END**;

Третий триггер:

Меняет статус возвращенного инвентаря на Свободен либо Сломан, если инвентарь вернулся в пункт проката поврежденным.

**CREATE** **TRIGGER** change\_inventory\_status\_after\_return **ON** Ski\_Base\_2021.dbo.InventoryReturn

**AFTER** **INSERT**

**AS** **BEGIN**

**IF** (**SELECT** IsDamaged **FROM** inserted) = 0 **UPDATE** dbo.Inventory **SET** Status = 'Available'

**WHERE** Inventory.InventoryId = (**SELECT** InventoryId **FROM** Rental **WHERE** Rental.RentalId = (**SELECT** RentalId **FROM** inserted))

**ELSE** **UPDATE** dbo.Inventory **SET** Status = 'Broken'

**WHERE** Inventory.InventoryId = (**SELECT** InventoryId **FROM** Rental **WHERE** Rental.RentalId = (**SELECT** RentalId **FROM** inserted))

**END**;

Первая хранимая процедура:

Приведенная ниже процедура написана для таблицы Rental. С ее помощью в данную таблицу можно вставлять новые записи, при этом будет происходить проверка, действительно ли человек, обслуживающий клиента в прокате, является работником проката (эта проверка может быть полезной, чтобы избежать случайных нежелательных опечаток), а также будет выбираться элемент инвентаря исходя из доступности (свободен ли на данный момент) инвентаря и пожеланий посетителя (какой именно инвентарь он хочет: лыжи, лыжные палки, шлем и т.д.

**CREATE** **PROCEDURE** dbo.insert\_note

@type\_name **VARCHAR**(20), @cid **INT**, @eid **INT**, @pos **VARCHAR**(20)

**AS** **BEGIN**

**DECLARE** @iid **INT**

**SET** @iid = (**SELECT** **TOP** 1 InventoryId

**FROM** dbo.joined\_tables

**WHERE** [Type] = @type\_name **AND** Status = 'Available')

**IF** (dbo.return\_employee\_position(@eid) **LIKE** @pos) **AND** @iid **NOT** **LIKE** ''

**INSERT** **INTO** Rental **VALUES**(@iid, @cid, @eid)

**ELSE** **PRINT** ('This employee is not from rental staff. Note will not be inserted in table.')

**END**

Можем сразу посмотреть на выполнение данной процедуры:

Изображение выглядит как текст

Автоматически созданное описание

Мы подали на вход идентификатор сотрудника 3, но сотрудник с таким идентификатором не работает в прокате, поэтому данная запись не добавилась в таблицу, вместо этого в выводе появилось сообщение.

На всякий случай проверим, что в таблице Аренда не добавилась запись:

Изображение выглядит как стол

Автоматически созданное описание

Попробуем выполнить ту же процедуру с теми же параметрами, поданными на вход, предварительно поменяв идентификатор сотрудника на правильный:

Изображение выглядит как текст

Автоматически созданное описание

Сообщение о невозможности вставки не появилось.

Появилась строка в таблице Аренда:

Изображение выглядит как стол

Автоматически созданное описание

Вторая хранимая процедура:

Эта процедура может пригодиться, если при заполнении данных произошла ошибка и предмету инвентаря с определенным уже назначенным ему идентификатором (суррогатным первичным ключом, который создается искусственно) присвоили неверный тип либо прописали его не в тот склад. Тогда приходит на помощь данная хранимая процедура, которой на вход нужно передать правильные параметры. Так как при первом появлении инвентаря на базе его еще никому не успевают сдать в аренду, по умолчанию при добавлении инвентаря в таблицу Инвентарь ему дается статус Свободен. Поэтому передавать хранимой процедуре на вход статус только что появившегося на базе элемента инвентаря бессмысленно.

**CREATE** **PROCEDURE** dbo.inventory\_update

@id **INT**, @tid **INT**, @houseid **INT**

**AS** **BEGIN**

**UPDATE** dbo.Inventory

**SET** TypeId = @tid, WarehouseId = @houseid

**WHERE** InventoryId = @id

**END**

## Список используемой литературы

1. Презентации семинаров С.М.Ямпольского за 3-4 модули
2. Презентации лекций А.Л.Бекларяна за 3-4 модули

## Заключение

В ходе написания работы по теме «Проектирование и разработка базы данных горнолыжной базы» под руководством С. М. Ямпольского были выполнены следующие задачи:

* Описание предметной области, т.е. процесс работы горнолыжной базы;
* Построение инфологической и даталогической моделей, описание сущностей и атрибутов предметной области, а также связей между этими сущностями;
* Реализация реляционной базы данных лыжной базы;
* Заполнение базы данных сведениями о сотрудниках, посетителях и их действиях;
* Разработка запросов к базе данных на языке SQL, а также создание пользовательских функции, триггеров, представлений и хранимых процедур.